Why configuration management is crucial

Anderson, P. (2006). Why configuration management is crucial. login;, 31(1).

PAUL ANDERSON

why configuration management is crucial

Paul Anderson has a background in pure mathematics and over 20 years of experience in system administration. He is currently a principal computing officer with the School of Informatics at Edinburgh University. He is the primary author of the LCFG configuration system and the organizer of the LISA configuration workshop series. His homepage is http://www.homepages.inf.ed.ac.uk/dcspaul/

dcspaul@inf.ed.ac.uk

THE PAST FEW YEARS HAVE SEEN an increasing interest in "Configuration Management." Some of us believe that the lack of good tools and procedures in this area is rapidly becoming *the* major barrier to the deployment of reliable, secure, and correct systems. I am going to try to define the configuration problem more clearly and to explore some possible reasons for these difficulties. However, it is worth starting with a few simple examples:

- Reliability—If we decommission a server, can we be certain that nothing else depends on this server in any way? Perhaps it might have been the only DHCP server on some little-used subnet, and problems will only become apparent when some host on that subnet fails to boot.
- Security—Can we be certain that the configuration files on a group of machines are set up so that there are no unexpected trust relationships between the machines? What if a supposedly secure machine installs new versions of an application from a remote file system on a less-secure server?
- Correctness—Can we be certain that every compute node in a cluster is running the required (new) version of a particular library, before starting a critical job?

An ideal configuration management system would prevent such problems by design. Current tools, used with best practice, should at least make it possible to identify and avoid them. However, sites with less-developed configuration management would even have difficulty in deciding whether or not such problems existed! For example, the information may only be available on the remote nodes themselves, and a certain percentage of these will always be unavailable at any one time.

Most sites have probably used some form of "configuration management" tool as a way of coping with large numbers of very similar "clients." This certainly addresses a whole class of configuration problems, such as the last of the above examples. However, modern computing installations form a complex web of related services. Managing the "servers" and the relationships they imply is much more difficult—this is the root cause of the deeper problems illustrated by the first two examples. The increasing scale, and particularly the complexity, of modern sites means that manual

approaches to configuration of these relationships are no longer adequate; human system administrators simply cannot manage the complexity of the interactions, or foresee the full consequences of individual configuration changes.

Achieving high reliability in complex systems also requires the capability for fully automated reconfiguration. An *autonomic* system must have the ability to reconfigure some other machine as replacement for a failed server. In a multi-tier Web service, for example, this is likely to involve extensive reconfiguration of related services.

What Is "System Configuration"?

The basic system configuration problem is quite simple to describe:

- Starting with:
  - A large number of varied machines with empty disks.
  - A repository of all the necessary software packages and data files.
  - A specification of the functions that the overall system is intended to perform.
- Load the software and configure the machines to provide the required functionality. This usually involves a good deal of internal infrastructure, e.g., DNS, LDAP, DHCP, NFS, NIS services.
- Reconfigure the machines whenever the required service specification changes.
- Reconfigure the machines to maintain conformance with the specification whenever the environment changes—for example, when things break.

In practice, the task of "configuring the machines" probably involves editing configuration files (or perhaps supplying the configuration information via some API or GUI). However, it is not the mechanics of this process that is important; the real difficulty is in determining a suitable configuration for each service on each host that will make the overall system behave according to the specification.

To solve difficulties such as those in the first two examples, a configuration system must have a model that can represent the relationships implied by the configuration of the individual machines. Conceptually, we can think of a configuration tool as a type of "compiler," whose input is a set of requirements for the entire system and whose output is a set of configuration parameters for each service on each host in the system. Of course, a real configuration tool involves a lot more practical details, such as formatting and distributing the configuration information, but these parts of the process are comparatively straightforward.

Ultimately, we would like the input language for our configuration compiler to be at a very high level. For example, we might specify a Web service with certain behavioral properties, and the compiler would generate the appropriate configurations for all of the individual services, on all of the participating hosts. Unlike most programming languages, these configuration specifications need to be *declarative*—i.e., we want to specify the required properties of the resulting configuration and have the tool automatically work out the procedures for achieving the end result.

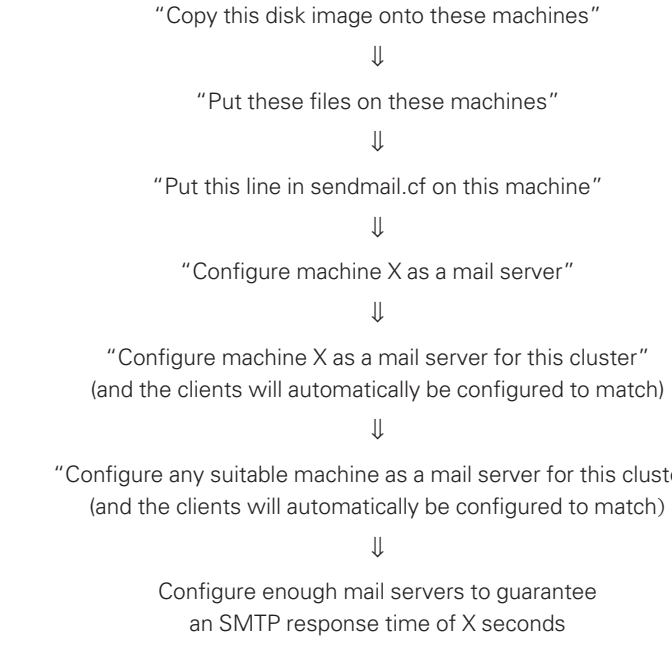

FIGURE 1

Present-day technology is some way from being able to translate such high-level specifications automatically; current best practice involves a combination of manual procedures and automatic tools that provide a smooth translation of the service requirements into implementable configuration details. Many factors, including the capabilities of the specific tool, will affect the level of detail at which the configuration needs to be manually specified. Figure 1 shows some possibilities, starting with very low-level tools which require all configuration decisions to be made manually, to very high-level tools which accept more abstract service requirements.

The final example defines a required behavior, and this is ultimately the type of specification that we would like to be able to make. However, this requires dynamic monitoring of performance levels, and the ability to do this in any general way is not yet part of any common configuration tool.

So What's the Problem?

The current situation with system configuration tools has many similarities with the early days of computer programming:

- Vendors sold mutually incompatible hardware. Changing platforms required a significant investment of time and resources.
- It was not possible to share code between machines without rewriting.
- The basic principles of programming had not yet been developed. Programs were created in unstructured ways that made them error-prone and difficult to verify or maintain.
- The low-level nature of the program code made it difficult to implement clear high-level objectives.

It was only the advent of high-level languages, with their underlying theory and portable compilers, that enabled this situation to improve.

In the system configuration field, there is a real need for new tools, based on sound theory and targeted at a much higher /level of configuration description. Programming-language development required a new genera-

tion of specialists to achieve a similar evolution, and it seems likely that real progress with configuration tools is not possible without a comparable development. This will require new specialists with a good understanding of theory, software development, and system administration practice.

As with programming-language development, the resulting systems will demand from working system administrators a significant change in approach. Worrying about which physical host is running a particular service should be as rare as worrying about which machine register is holding a particular Java variable!